TESTES EM ANDROID COM ESPRESSO

CONSIDERAÇÕES

- Apresentar a ferramenta ao time de QA/Solution BS2
- Vou terminar a apresentação sabendo automatizar? Não
- Por quê?

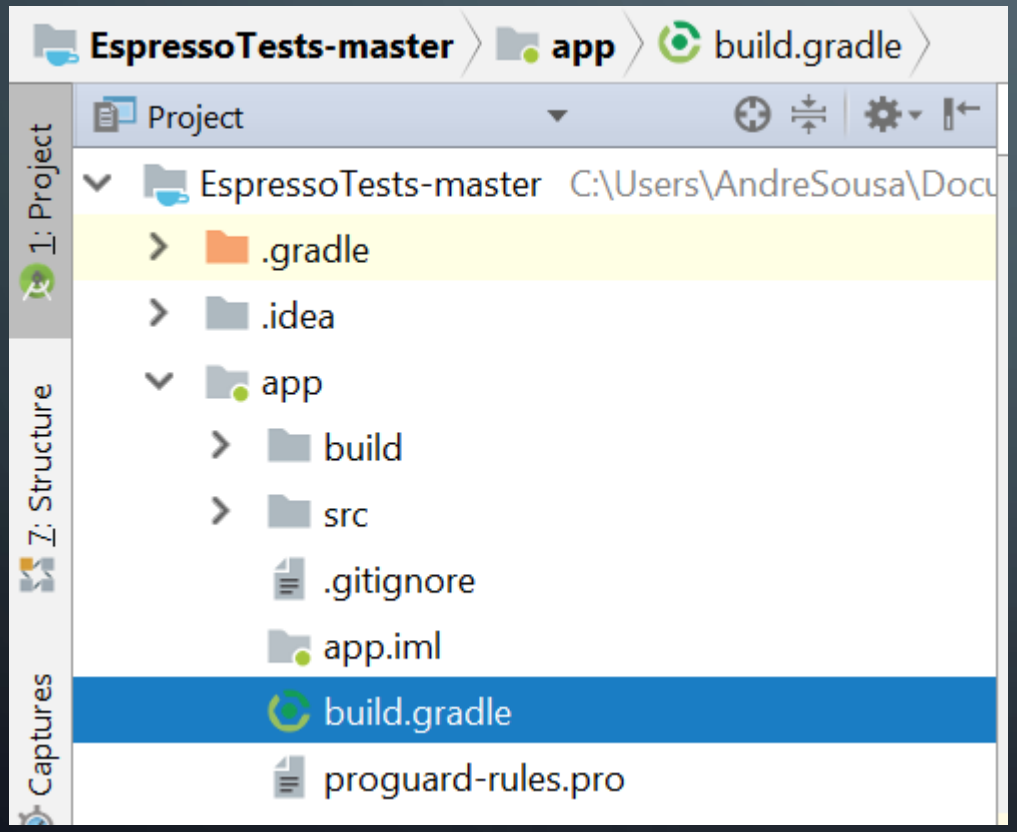
CONFERINDO

- Download do projeto(App) para teste disponibilizado
- Android Studio instalado
- Celular android ou dispositivo virtual

COMO REALIZAR TESTES EM UM APP ANDROID UTILIZANDO O ESPRESSO

- O app base chama EspressoTest
- Possui 3 telas(Login, Lista de Usuários, Detalhes do Usuário)

CONFIGURAÇÕES DE BIBLIOTECAS

- 

```
androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
androidTestImplementation 'com.android.support.test:runner:1.0.2'
androidTestImplementation 'com.android.support.test:rules:1.0.2'
androidTestImplementation 'com.android.support.test.espresso:espresso-intents:3.0.2'
implementation 'com.android.support:support-annotations:27.1.1'
androidTestImplementation 'junit:junit:4.12'
```

JUNIT

- JUnit é um framework open-source, com suporte à criação de testes automatizados na linguagem de programação Java.
- Pode-se criar uma hierarquia de testes que permitirá testar apenas uma parte do sistema ou todo ele.
- Exemplos de parâmetros de hierarquia - @Before, @Rules, @Test, @After, etc.
- Assim como o Junit, existem outros frameworks bem comuns que facilitam as validações, como o Hamcrest e UI Automator.

CRIANDO O PRIMEIRO TESTE

- EspressoTests-master/app/src/androidTest/
  - Arquivo: TestCase1
- @Test

```
public void quandoActivityIniciada_visualizarEstadoTela () {
  onView(withId(R.id.login_image)).check(matches(isDisplayed()));
  onView(withId(R.id.login_username)).check(matches(isDisplayed()));
  onView(withId(R.id.login_password)).check(matches(isDisplayed()));
  onView(withId(R.id.login_button)).check(matches(isDisplayed()));
}
```

Uma coleção de objetos que implementam a Matcher<? super View>interface.

ONVIEW(WITHID(R.ID.LOGIN_IMAGE)).CHECK(MATCHES(ISDISPLAYED()));

- **ViewMatchers** - Uma coleção de objetos que utiliza uma ou mais asserções dentro do teste.
- onView - Ponto de entrada para interações.
- withId – recebe o id do componente a ser realizada a interação.
- .check - vai verificar se é válida a declaração que estamos passando como parâmetro.
- Matches – funciona como um assert, utilizando a validação parametrizada.

EXECUTANDO

- Vá em *Configurações > Programador (ou opções de desenvolvedor)* e desative estas três opções:
- Animação em escala;
- Escala de transição;
- Escala de duração da animação.

- Deve ser feito para o device ou simulador.
- O Espresso aguarda a UI Thread ficar ociosa (idle) para executar o próximo passo do teste. Porém, se as animações estiverem ligadas, ele irá se perder e os testes irão falhar.

CONFIRMANDO A EXECUÇÃO ATRAVÉS DA NEGAÇÃO

- Adicionar a biblioteca
- import static org.hamcrest.Matchers.not;

- Alterar a linha
- onView(withId(R.id.login_image)).check(matches(not(isDisplayed())));

RESULTADO

- android.support.test.espresso.base.DefaultFailureHandler$AssertionFailedWithCause Error: **'not is displayed on the screen to the user' doesn't match the selected view.**
- **Expected:** not is displayed on the screen to the user
- **Got:** "AppCompatImageView{id=2131165258, res-name=login_image, desc=Android image, ***visibility=VISIBLE***, width=144, height=144, has-focus=false, has-focusable=false, has-window-focus=true, is-clickable=false, is-enabled=true, is-focused=false, is-focusable=false, is-layout-requested=false, is-selected=false, layout-params=android.widget.RelativeLayout$LayoutParams@865b0ae, tag=null, root-is-layout-requested=false, has-input-connection=false, x=468.0, y=508.0}"

PREENCHENDO CAMPOS E EXECUTANDO AÇÕES

- Cenário 1: Validar exibição de modal com um dos campos(user/password) não preenchidos. Arquivo: TestCase2
  - utilizamos novos métodos:
    - perform(): vai executar a ViewAction que ele recebe como parâmetro;
    - typeText(): ViewAction para digitar um texto na view;
    - click(): ViewAction para clicar na view;
    - withText(): vai procurar a view que contenha o texto passado como parâmetro.

- O nosso bloco de teste está executando as seguintes ações:
- Escrevendo um texto no campo *login_username* ou *login_password*;
- Clicando no botão de login;
- Verificando que o pop-up está aparecendo (pois deixamos um dos campos em branco);
- Clicando no *Ok* do dialog para fechá-lo.

REFATORANDO OS TESTES

- **1° refatoração:** Dividindo os testes para ficarem independentes;
  - Dessa forma, um possível erro no campo login não interfere na validação do campo senha e vice-versa.
- **2° refatoração:** Tornando o código reutilizável
  - Associamos o campo ao método/teste e criamos um preenchimento default já que seu conteúdo não interfere na validação do diálogo.

APLICANDO UM LOGIN VÁLIDO AOS CENÁRIOS ANTERIORES

- @Test

```
public void quandoCamposPreenchidos_deveAbrirMainActivity() {
  onView(withId(R.id.login_username)).perform(typeText("defaultText"), closeSoftKeyboard());
  onView(withId(R.id.login_password)).perform(typeText("defaultText"), closeSoftKeyboard());
  onView(withId(R.id.login_button)).perform(click());
  onView(withId(R.id.main_activity_container)).check(matches(isDisplayed()));
}
```

Funciona, mas tá errado.

POR QUÊ????

MOTIVOS

1. Evite fazer testes que façam navegações através do app, inicie as activities diretamente no estado desejado.
2. A MainActivity faz uma requisição para a API, o que torna este teste dependente de um recurso externo.

Uma boa prática é deixar os testes isolados de qualquer dependência externa.

COMO FAZER ENTÃO?

- Configurar um IntentsTestRule;

```
private void doLogin() {
  startActivity(new Intent(this, MainActivity.class));
}
```

- A intent que estamos montando é bem simples, pois possui apenas o nome da classe (MainActivity.class) e o contexto como parâmetros.

TESTANDO O LOGIN

```
@Test
public void quandoCamposPreenchidos_deveAbrirMainActivity() {
  Intents.init();  -- Estamos iniciando a gravação das intents com o método init();
  onView(withId(R.id.login_username)).perform(typeText("username"), closeSoftKeyboard());
  onView(withId(R.id.login_password)).perform(typeText("password"), closeSoftKeyboard());
  Matcher<Intent> matcher = hasComponent(MainActivity.class.getName());  -- Usamos o
método IntentMatchers.hasComponent(String className) passando como parâmetro o nome da classe MainActivity, que
é a activity que será iniciada
  onView(withId(R.id.login_button)).perform(click());
  intended(matcher); --O método intended(Matcher<Intent> matcher) verifica que o matcher passado como parâmetro
é o que a activity em teste irá lançar, garantindo também que esta intent seja única;
  Intents.release(); -- limpa o estado das intents
}
```

ISOLANDO O TESTE

- Garantindo que o teste esteja isolado e evitando a navegação pelo app;

```
ActivityResult result = new ActivityResult(Activity.RESULT_OK, null);
 intending(matcher).respondWith(result);
```

Com esse novo trecho de código, fizemos o seguinte:

-- criamos um objeto ActivityResult que irá simular o resultado da activity;

-- Usamos o método intending() para devolver um resultado assim que a intent for lançada.

-- Chamamos o método respondWith(), passando como parâmetro o nosso objeto ActivityResult (result)

CONTEÚDO

- Esse documento foi criado utilizando a documentação oficial do Android e encontra-se disponível em:
- https://developer.android.com/training/testing/espresso/